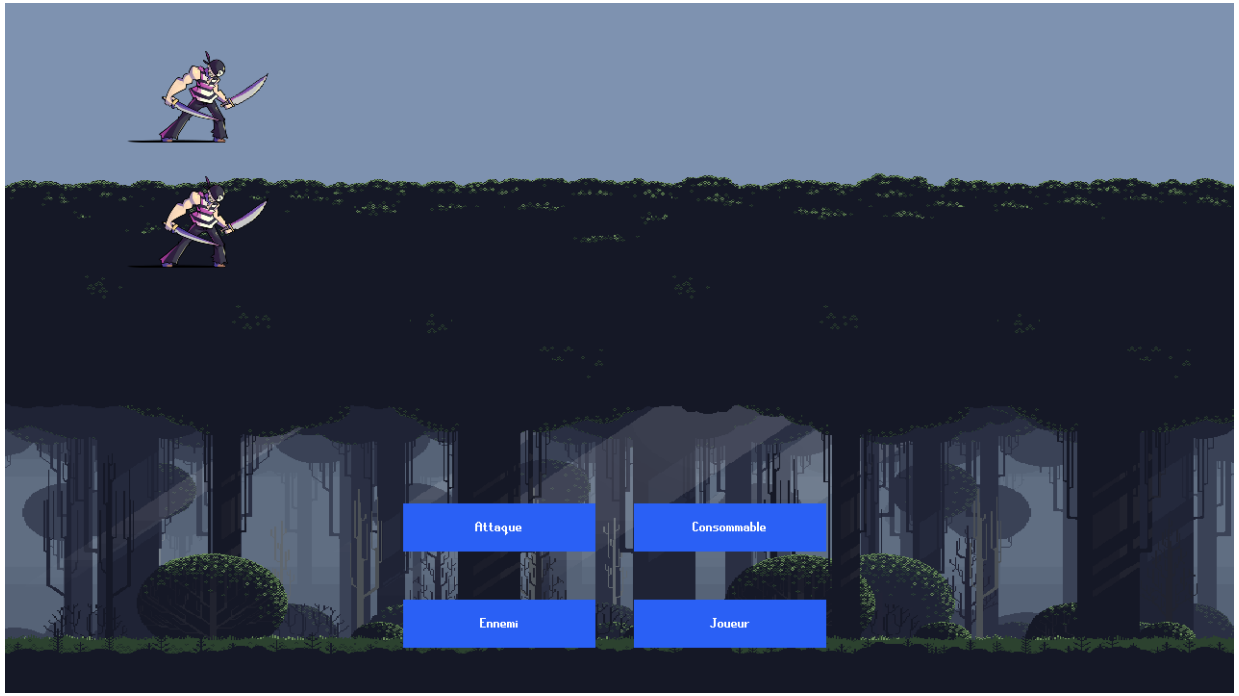
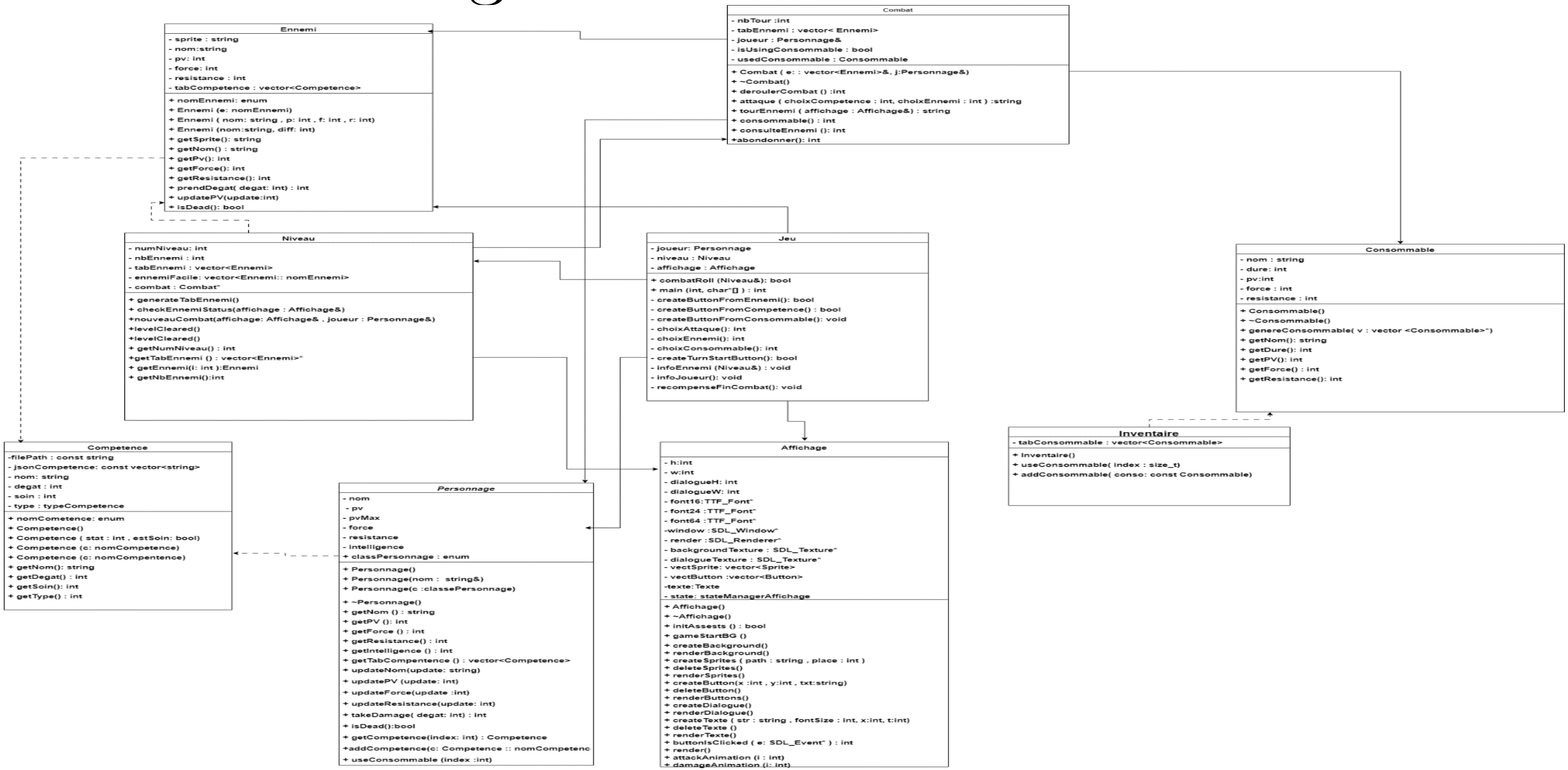
RANDOM COMBAT

"RandomCombat" est un jeu basé sur des combats aléatoires entre un personnage joueur et différents ennemis. Le jeu utilise la bibliothèque SDL (Simple DirectMedia Layer) pour la gestion des graphismes et des événements. L'objectif du joueur est de survivre à un certain nombre de combats successifs, chaque combat devenant progressivement plus difficile.



PROJET DE LIFAPCD FAIT PAR Chahd Aloui et Jules Coasne

Diagramme des classes



1. Difficultés et classes intéressantes

Gérer plusieurs textures :

Dans mon code, nous avons utilisé des textures pour représenter différents éléments graphiques du jeu, tels que les personnages, les boutons et les arrière-plans. La gestion de plusieurs textures peut être complexe car elle nécessite de charger, de manipuler et de libérer correctement la mémoire des textures pour éviter les erreurs d'affichage. Il est important de gérer soigneusement les cycles de vie des textures et de s'assurer qu'elles sont chargées et déchargées de manière appropriée.

```
void Affichage::gameStartBG(){
    SDL_Surface * s = SDL_CreateRGBSurface(0, w, h, 32, 0, 0, 0, 0);
    if(s == NULL){
        cout << "Erreur lors de l'initialisation de la surface startBG" << SDL_GetError() << endl;
    }
    SDL_SetSurfaceBlendMode(s, SDL_BLENDMODE_BLEND);

    SDL_Texture * t = SDL_CreateTextureFromSurface(renderer, s);
    if(t == NULL){
        cout << "Erreur lors de l'initialisation de la texture startBG" << SDL_GetError() << endl;
    }


    SDL_RenderCopy(renderer, t, NULL, NULL);
    SDL_FreeSurface(s);
}

void Affichage::createBackground(){
    SDL_Surface *background_surface = IMG_Load("data/bg/background.png");
    if(background_surface == NULL){
        cout << "Erreur chargement du terrain" << SDL_GetError() << endl;
    }

    backgroundTexture = SDL_CreateTextureFromSurface(renderer, background_surface);
    SDL_FreeSurface(background_surface);
}
```

2. Difficultés et classes intéressantes

Les boutons :

Les boutons sont des éléments essentiels de l'interface utilisateur de votre jeu, permettant au joueur d'effectuer des actions telles que sélectionner une compétence ou naviguer dans les menus. La difficulté réside dans la création, le positionnement et l'interaction avec les boutons à l'écran. Il est nécessaire de détecter les clics de souris sur les boutons, de gérer les états des boutons (par exemple, survolé, enfoncé) et de déclencher les actions associées lorsque les boutons sont activés. Une bonne gestion des boutons garantit une expérience utilisateur fluide et intuitive.

```
void Affichage::renderButtons(){
    for(size_t i = 0; i < vectButton.size(); i++){
        SDL_Surface * surface = TTF_RenderText_Solid(font16, vectButton[i].texte, SDL_Color {255, 255, 255});
        SDL_Texture * texture = SDL_CreateTextureFromSurface(renderer, surface);

        SDL_Rect txtRect;
        txtRect.w = surface->w;
        txtRect.h = surface->h;
        txtRect.x = vectButton[i].descRect.x + (vectButton[i].descRect.w - txtRect.w) / 2;
        txtRect.y = vectButton[i].descRect.y + (vectButton[i].descRect.h - txtRect.h) / 2;

        SDL_RenderCopy(renderer, vectButton[i].texture, NULL, &vectButton[i].descRect);
        SDL_RenderCopy(renderer, texture, NULL, &txtRect);

        SDL_FreeSurface(surface);
    }
}
```

3. Difficultés / classes intéressantes

Classe Ennemi :

Encapsulation des données : La classe Ennemi encapsule les données relatives à un ennemi spécifique, telles que son nom, ses points de vie, sa force, sa résistance, etc. Cela permet de regrouper toutes les informations concernant un ennemi en un seul objet, facilitant ainsi la manipulation et la gestion des ennemis dans votre jeu.

Flexibilité : Grâce aux différents constructeurs de la classe Ennemi, vous pouvez créer des ennemis de manière flexible en chargeant leurs données à partir de fichiers JSON, en spécifiant directement leurs attributs ou en les ajustant en fonction d'un niveau de difficulté. Cela offre une grande flexibilité dans la conception et la création d'ennemis variés pour votre jeu.

Modularité : La classe Ennemi est conçue de manière modulaire, avec des méthodes telles que prendDegat() pour gérer les dégâts reçus par l'ennemi et isDead() pour vérifier si l'ennemi est mort. Cette modularité facilite l'extension et la maintenance du code, permettant d'ajouter facilement de nouvelles fonctionnalités ou de modifier le comportement des ennemis au besoin.

4. Difficultés / classes intéressantes

Classe Personnage :

Personnalisation du personnage : La classe Personnage permet de créer et de personnaliser le personnage jouable du joueur en définissant des attributs tels que le nom, les points de vie, la force, l'intelligence, etc. Cela offre une expérience de jeu plus immersive en permettant aux joueurs de créer leur propre personnage avec des caractéristiques uniques.

Gestion des compétences : La classe Personnage gère les compétences du personnage jouable à travers le vector tabCompetence. Cela permet d'ajouter, de supprimer ou de modifier facilement les compétences du personnage, offrant ainsi une grande variété de styles de jeu et de stratégies possibles.

Inventaire de consommables : La classe Personnage gère également l'inventaire de consommables du personnage à travers le vector tabConsommable. Cela permet aux joueurs d'accumuler et d'utiliser des objets de soin ou des boosts pendant le jeu, ce qui ajoute une dimension stratégique supplémentaire à l'expérience de jeu.

Conclusion

- Toutes les fonctionnalités qu'on avait prévues au départ dans notre cahier des charges sont désormais intégrées dans le jeu, ce qui est une grande réussite pour nous. On a réussi à créer ce système de combat qui nous tenait à cœur, à mettre en place une interface utilisateur sympa avec des animations, et même à inclure des fonctionnalités comme le système de score. Mais bien sûr, il y a toujours cette envie d'en faire plus. Si on avait eu un peu plus de temps, on aurait aimé ajouter des animations pour rendre les combats encore plus vivants. On aurait peut-être aussi pensé à ajouter d'autres petites choses pour rendre le jeu encore plus amusant, comme des quêtes secondaires ou des événements aléatoires. Enfin, il y a toujours des ajustements à faire pour rendre le jeu encore plus équilibré et l'interface encore plus facile à utiliser.